Chapter 2

A Review of Database System Terminology

Marion G. Ceruti

MANY PUBLICATIONS, TECHNICAL MANUALS, AND MARKETING BROCHURES related to databases originated from sources that exhibit a wide variety of training, background, and experience. Although the result has been an expanded technical vocabulary, the growth of standards — particularly with regard to a comprehensive, uniformly accepted terminology — has not kept pace with the growth in the technology itself. Consequently, the nomenclature used to describe various aspects of database technology is characterized, in some cases, by confusion and chaos. This is true for both homogeneous databases and for heterogeneous, distributed database systems.

The state of imprecision in the nomenclature of this field persists across virtually all data models and their implementations. The purpose of this chapter is to highlight some areas of conflict and ambiguity and, in some cases, to suggest a more meaningful use of the terminology.

GENERAL DATABASE TERMS

What Does the Word *Data* Mean?

According to Webster, the word *data* is a noun that refers to things known or assumed; facts or figures from which conclusions can be inferred; information. Derived from the Latin word *datum*, meaning gift or present, data can be given, granted, or admitted, premises upon which something can be argued or inferred. Although the word *data* is most frequently observed, the singular form, *datum*, is also a real or assumed thing used as the basis for calculations.

The Department of Defense defines data as a representation of facts, concepts, or instructions in a formalized manner suitable for communication, interpretation, or processing by humans or by automatic means.

The word *data* is also used as an adjective in terms such as *data set, data fill, data resource, data management,* or *data mining*. A data set is an aggregate of data items that are interrelated in some way.

Implicit in both definitions of data is the notion that the user can reasonably expect data to be true and accurate. For example, a data set is assumed to consist of facts given for use in a calculation or an argument, for drawing a conclusion, or as instructions from a superior authority. This also implies that the data management community has a responsibility to ensure the accuracy, consistency, and currency of data.

Data Element vs. Data Item

In an attempt to define database terms with a view toward practical applications, the Department of Defense (DoD) defines a data element as a named identifier of each of the entities and their attributes that are represented in a database. As such, data elements must be designed as follows:

- Representing the attributes (characteristics) of data entities identified in data models.
- According to functional requirements and logical (as opposed to physical) characteristics.
- According to the purpose or function of the data element, rather than how, when, where, and by whom it is used.
- With singularity of purpose, such that it has only one meaning.
- With well-defined, unambiguous, and separate domains.

Other definitions are that a data element is data described at the useful primitive level; a data item is the smallest separable unit recognized by the database representing a real-world entity.

What is clear from all these definitions is that there is considerable ambiguity in what these terms mean. The author proposes the following distinction between data element and data item:

> A *data element* is a variable associated with a domain (in the relational model) or an object class (in the object-oriented model) characterized by the property of atomicity. A data element represents the smallest unit of information at the finest level of granularity present in the database. An instance of this variable is a *data item*. A data element in the relational model is simply an attribute (or column) that is filled by data items commonly called the "data fill."

This distinction clarifies but does not preclude any of the other definitions.

What Is a Database?

The definitions for the term *database* range from the theoretical and general to the implementation specific. For example, K.S. Brathwaite, H. Darwen, and C.J. Date have offered two different, but not necessarily inconsistent, definitions of a database that are specific to the relational model. Darwen and Date build their definition on fundamental constructs of the relational model, and it is very specific to that model. Brathwaite employs a definition that is based on how databases are constructed in a specific database management system (DBMS).

These definitions are discussed in the next section on relational database terms. Actually, the term *database* can have multiple definitions, depending on the level of abstraction under consideration. For example, A.P. Sheth and J.A. Larson define database in terms of a reference architecture, in which a database is a repository of data structured according to a data model. This definition is more general than that of either Brathwaite or Darwen and Date because it is independent of any specific data model or DBMS. It could apply to hierarchical and object- oriented databases as well as to relational databases; however, it is not as rigorous as Darwen and Date's definition of a relational database because the term *repository* is not defined.

Similarly, P.J. Fortier et al., in a set of DoD conference proceedings, define a database to be a collection of data items that have constraints, relationships, and a schema. Of all the definitions for database considered thus far, this one is the one most similar to that of Sheth and Larson, because the term *data model* could imply the existence of constraints, relationships, and a schema. Moreover, Fortier et al. define *schema* as a description of how data, relationships, and constraints are organized for user application program access. A *constraint* is a predicate that defines all correct states of the database. Implicit in the definition of schema is the idea that different schemata could exist for different user applications. This notion is consistent with the concept of multiple schemata in a federated database system (FDBS). (Terms germane to FDBSs are discussed in a subsequent section.)

L.S. Waldron defines *database* as a collection of interrelated files stored together, where specific data items can be retrieved for various applications. A file is defined as a collection of related records. Similarly, L. Wheeler defines a *database* as a collection of data arranged in groups for access and storage; a database consists of data, memo, and index files.

Database System vs. Data Repository

Both of these terms refer to a more comprehensive environment than a database because they are concerned with the tools necessary for the management of data in addition to the data themselves. These terms are not mutually exclusive. A *database system* (DBS) includes both the DBMS software and one or more databases. A *data repository* is the heart of a comprehensive information management system environment. It must include not only data elements, but metadata of interest to the enterprise, data screens, reports, programs, and systems.

A data repository must provide a set of standard entities and allow for the creation of new, unique entities of interest to the organization. A database system can also be a data repository that can include a single database or several databases.

A. King et al. describe characteristics of a data repository as including an internal set of software tools, a DBMS, a metamodel, populated metadata, and loading and retrieval software for accessing repository data.

WHAT IS A DATA WAREHOUSE AND WHAT IS DATA MINING?

B. Thuraisingham and M. Wysong discussed the importance of the data warehouse in a DoD conference proceeding. A *data warehouse* is a database system that is optimized for the storage of aggregated and summarized data across the entire range of operational and tactical enterprise activities. The data warehouse brings together several heterogeneous databases from diverse sources in the same environment. For example, this aggregation could include data from current systems, legacy sources, historical archives, and other external sources.

Unlike databases that are optimized for rapid retrieval of information during real-time transaction processing for tactical purposes, data warehouses are not updated, nor is information deleted. Rather, time-stamped versions of various data sets are stored. Data warehouses also contain information such as summary reports and data aggregates tailored for use by specific applications. Thus, the role of metadata is of critical importance in extracting, mapping, and processing data to be included in the warehouse. All of this serves to simplify queries for the users, who query the data warehouse in a read-only, integrated environment.

The data warehouse is designed to facilitate the strategic, analytical, and decision-support functions within an organization. One such function is *data mining*, which is the search for previously unknown information in a data warehouse or database containing large quantities of data. The data warehouse or database is analogous to a mine, and the information desired is analogous to a mineral or precious metal.

The concept of data mining implies that the data warehouse in which the search takes place contains a large quantity of unrelated data and probably was not designed to store and support efficient access to the information desired. In data mining, it is reasonable to expect that multiple, well-designed queries and a certain amount of data analysis and processing will be necessary to summarize and present the information in an acceptable format.

Data Administrator vs. Database Administrator

The following discussion is not intended to offer an exhaustive list of tasks performed by either the data administrator (DA) or database administrator (DBA), but rather to highlight the similarities and essential distinctions between these two types of database professionals. Both data administrators and database administrators are concerned with the management of data, but at different levels.

The job of a *data administrator* is to set policy about determining the data an organization requires to support the processes of that organization. The data administrator develops or uses a data model and selects the data sets supported in the database. A data administrator collects, stores, and disseminates data as a globally administered and standardized resource. Data standards on all levels that affect the organization fall under the purview of the data administrator, who is truly an administrator in the managerial sense.

By contrast, the technical orientation of the *database administrator* is at a finer level of granularity than that of a data administrator. For this reason, in very large organizations, DBAs focus solely on a subset of the organization's users. Typically, the database administrator is, like a computer systems manager, charged with day-to-day, hands-on use of the DBS and daily interaction with its users. The database administrator is familiar with the details of implementing and tuning a specific DBMS or a group of DBMSs. For example, the database administrator has the task of creating new user accounts, programming the software to implement a set of access controls, and using audit functions.

To illustrate the distinction between a data administrator and a database administrator, the U.S. Navy has a head data administrator whose range of authority extends throughout the entire Navy. It would not be practical or possible for an organization as large as the U.S. Navy to have a database administrator in an analogous role, because of the multiplicity of DBSs and DBMSs in use and the functions that DBAs perform.

These conceptual differences notwithstanding, in smaller organizations a single individual can act as both data administrator and database administrator, thus blurring the distinction between these two roles. Moreover, as data models and standards increase in complexity, data administrators will increasingly rely on new technology to accomplish their tasks, just as database administrators do now.

RELATIONAL DATABASE TERMS

Because relational technology is a mature technology with many practical applications, it is useful to consider some of the important terms that pertain to the relational model. Many of these terms are straightforward and generally unambiguous, whereas some terms have specific definitions that are not always understood.

A data set represented in the form of a table containing columns and rows is called a *relation*. The columns are called *attributes*, and the rows are called *tuples*.

Darwen and Date define a tuple to be a set of ordered triples of the form <A, V, v> where A is the name of an attribute, V is the name of a unique domain that corresponds to A, and v is a value from domain V called the attribute value for attribute A within the tuple. A *domain* is a named set of values.

Darwen and Date also describe a relation as consisting of a heading and a body, where the heading is a set of ordered pairs, <A,V>; and the body consists of tuples, all having the same heading <A,V>. An *attribute value* is a data item or a datum.

In some respects, a relation is analogous to an array of data created outside a relational DBMS, such as in a third-generation language (3GL) program like C, FORTRAN, or Ada, in which the rows are called records and the columns are called fields. Waldron defines a *field* as a set of related letters, numbers, or other special characters, and defines a *record* as a collection of related fields.

The interchangeability of the terms *record* and *row* has been illustrated by some of the major DBMS vendors in the way in which they report the results of a query to the user. Earlier versions of commercial DBMSs indicated at the end of a query return messages such as "12 records selected." Now, it is more common to see messages such as "12 rows selected" or "12 rows affected" instead.

Relation vs. Relation Variable

The correct manner in which the term *relation* should be used is according to the definition given previously, which specifically includes values *v*, from domain V. However, the term *relation* has not always been used correctly in the industry. Relation frequently is used as though it could mean either a filled table with data present (correct), or an empty table structure containing only data headers (incorrect). The confusion here stems from a failure to distinguish between a *relation*, which is a filled table with tuples containing attribute values, and a *relation variable* (or relvar), which is an empty table structure with only attribute names and domains from which to choose values. The values of a relation variable are the relations per se. This distinction becomes especially important when mapping between the relational and object-oriented data models.

Database vs. Database Variable

In a manner similar to the relation-relvar dichotomy, a *database variable* is different from a database per se. A database variable (or dbvar) is a named set of relvars. The value of a given dbvar is a set of specific, ordered pairs <R,r>, where R is a relvar and r (a relation) is the current value of that relvar, such that one such ordered pair exists for each relvar in the dbvar and that, taken together, all relvar values satisfy the applicable constraints (in particular, integrity constraints). A value of the dbvar that conforms to this definition is called a database. Some call this a *database state*, but this term is not used very often.

Database vs. DBMS

As all the examples discussed thus far indicate, not all database terminology is as unambiguous as "rows" and "columns." Incorrect understanding of the fundamental concepts in database technology can lead to inconsistent terminology, and vice versa.

DBMS Software Does Not Equal a Database. For example, databases frequently are described according to the DBMS that manages them. This is all well and good, as long as one realizes that references to an Oracle database and Sybase database refer to the databases that are managed using Oracle or Sybase software, respectively. Difficulty arises when this nomenclature results in the misconception that DBMS software is actually the database itself. The assumption that Informix, for example, is a database is as illogical as thinking that the glass is the same as the water in it.

Concept vs. Implementation in Relational Databases

Darwen and Date's definition of a database, as well as that of other database researchers (some of whom are mentioned by name in this chapter and others who are not), does not require the presence of a DBMS. Conceptually, it is possible to have a database without a DBMS or a DBMS without a database, although obviously the greatest utility is achieved by combining the two.

In the context of a specific DBMS environment, Brathwaite defines an IBM DB2 database as "a collection of table and index spaces where each table space can contain one or more physical tables." This definition is inconsistent with Date's definition because it allows for the possibility that the table spaces could be empty, in which case no data would be present. It is not clear that even relvars would be present in this case. That notwithstanding, if physical tables are present, Brathwaite's definition becomes an implementation-specific special case of Date's definition. (Substitute the word "must" for "can" to resolve the problem with Brathwaite's definition.)

Except in the case where the vendor has specified default table and index spaces in the DBMS code, the database and index spaces are not actually part of the DBMS per se. The DBA needs to create both the database space and the index space using the DBMS software.

DATABASE NORMALIZATION

The topic of *database normalization*, sometimes called *data normalization*, has received a great deal of attention. As is usually the case, database normalization is discussed in the following section using examples from the relational data model. Here, the terms *relation* and *table* are used interchangeably. However, the design guidelines pertaining to database normalization are useful even if a relational database system is not used. For example, B.S. Lee has discussed the need for normalization in the object-oriented data model. Whereas the intent of this section is to introduce the correct usage of normalization terminology as it applies to database technology, it is not meant to be an exhaustive exposition of all aspects of normalization.

What Is Database Normalization?

Strictly speaking, database normalization is the arrangement of data into tables. P. Winsberg defines normalization as the process of structuring data into a tabular format, with the implicit assumption that the result must be in at least first normal form. Similarly, Brathwaite defines data normalization as a set of rules and techniques concerned with:

- Identifying relationships between attributes
- Combining attributes to form relations (with data fill)
- Combining relations to form a database

The chief advantage of database or data normalization is to avoid modification anomalies that occur when facts about attributes are lost during insert, update, and delete transactions. However, if the normalization process has not progressed beyond first normal form, it is not possible to ensure that these anomalies can be avoided. Therefore, database normalization commonly refers to further non-loss decomposition of the tables into second through fifth normal form. Non-loss decomposition means that information is not lost when a table in lower normal form is divided (according to attributes) into tables that result in the achievement of a higher normal form. This is accomplished by placing primary and foreign keys into the resulting tables so that tables can be joined to retrieve the original information.

What Are Normal Forms?

A normal form of a table or database is an arrangement or grouping of data that meets specific requirements of logical design, key structure, modification integrity, and redundancy avoidance, according to the rigorous definition of the normalization level in question. A table is said to be in "X" normal form if it is already in "X-1" normal form and it meets the additional constraints that pertain to level "X."

In first normal form (1NF), related attributes are organized into separate tables, each with a primary key. A primary key is an attribute or set of attributes that uniquely defines a tuple. Thus, if a table is in 1NF, entities within the data model contain no attributes that repeat as groups. W. Kent has explained that in 1NF, all occurrences of a record must contain the same number of fields. In 1NF, each data cell (defined by a specific tuple and attribute) in the table will contain only atomic values.

Every table that is in second normal form (2NF) also must be in 1NF, and every non-key attribute must depend on the entire primary key. Any attributes that do not depend on the entire key are placed in a separate table to preserve the information they represent. 2NF becomes an issue only for tables with composite keys. A composite key is defined as any key (candidate, primary, alternate, or foreign) that consists of two or more attributes. If only part of the composite key is sufficient to determine the value of a non-key attribute, the table is not in 2NF.

Every relation that is in third normal form (3NF) must also be in 2NF, and every non-key attribute must depend directly on the entire primary key. In 2NF, non-key attributes are allowed to depend on each other. This is not allowed in 3NF. If a non-key attribute does not depend on the key directly, or if it depends on another non-key attribute, it is removed and placed in a new table. It is often stated that in 3NF, every non-key attribute is a function of "the key, the whole key, and nothing but the key." In 3NF, every non-key attribute must contribute to the description of the key. However, 3NF does not prevent part of a composite primary key from depending on a non-key attribute, nor does it address the issue of candidate keys.

Boyce-Codd normal form (BCNF) is a stronger, improved version of 3NF. Every relation that is in BCNF also must be in 3NF and must meet the additional requirement that each determinant must be a candidate key. A determinant is any attribute, A, of a table that contains unique data values, such that the value of another attribute, B, fully functionally depends on the value of A. If a candidate key also is a composite key, each attribute in the composite key must be necessary and sufficient for uniqueness. Winsberg calls this condition "unique and minimal." Primary keys meet these requirements. An alternate key is any candidate key that is not the primary key. In BCNF, no part of the key is allowed to depend on any key attribute. Compliance with the rules of BCNF forces the database designer to store associations between determinants in a separate table, if these determinants do not qualify as candidate keys.

BCNF removes all redundancy due to singular relationships but not redundancy due to many-to-many relationships. To accomplish this, further normalization is required. Fourth and fifth normal forms (4NF and 5NF) involve the notions of multivalued dependence and cyclic dependence, respectively. A table is in 4NF if it also is in BCNF and does not contain any independent many-to-many relationships.

That notwithstanding, a table could be in 4NF and still contain dependent many-to-many relationships. A table is in 5NF if it is also in 4NF and does not contain any cyclic dependence (except for the trivial one between candidate keys.) In theory, 5NF is necessary to preclude certain join anomalies, such as the introduction of a false tuple. However, in practice, the large majority of tables in operational databases do not contain attributes with cyclical dependence.

What Are Over-Normalization and Denormalization?

Over-normalization of a table results in further non-loss decomposition that exceeds the requirements to achieve 5NF. The purpose of this is to improve update performance. However, most operational databases rarely reach a state in which the structure of all tables has been tested according to 5FN criteria, so over-normalization rarely occurs. Over-normalization is the opposite of denormalization, which is the result of intentionally introducing redundancy into a database design to improve retrieval performance. Here, the database design process has progressed to 3NF, BCNF, 4NF, or even to 5NF. However, the database is implemented in a lower normal form to avoid time-consuming joins. Because the efficiency of "select" queries is an issue in operational systems, denormalization is more common than over-normalization.

The first six normal forms (including BCNF) are formal structures of tables that eliminate certain kinds of intra-table redundancy. For example, 5NF eliminates all redundancy that can be removed by dividing tables according to attributes. Higher normal forms exist beyond 5NF. They address theoretical issues that are not considered to be of much practical importance. In fact, Date has noted that it is not often necessary or desirable to carry out the normalization process too far because normalization optimizes update performance at the expense of retrieval performance. Most of the time, 3NF is sufficient. This is because tables that have been designed logically and correctly in 3NF are almost automatically in 4NF. Thus, for most databases that support real-time operations, especially for those that have tables with predominantly single-attribute primary keys, 3NF is the practical limit. Note that a two-attribute relation with a single-attribute key is automatically in the higher normal forms.

DISTRIBUTED, HETEROGENEOUS DATABASE NOMENCLATURE

What Is a Distributed Database?

Date defines a distributed database as a virtual database that has components physically stored in a number of distinct "real" databases at a number of distinct sites.

Federated Database Systems vs. Multidatabase Systems. M. Hammer and D. McLeod coined the term *federated database system* to mean a collection of independent, preexisting databases for which data administrators and database administrators agree to cooperate. Thus, the database administrator for each component database would provide the federation with a schema representing the data from his or her component that can be shared with other members of the federation.

In a landmark paper ("Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases," *ACM Computing Surveys*, Vol. 22, No. 3, September 1990), Sheth and Larson define FDBS in a similar but broader architectural sense to mean a collection of cooperating but autonomous component database systems that are possibly heterogeneous. They also define a *nonfederated database* system as an integration of component DBMSs that is not autonomous with only one level of management, in which local and global users are not distinguished. According to Sheth and Larson's taxonomy, both federated and nonfederated database systems are included in a more general category called *multidatabase systems*. These multidatabase systems support operations on multiple-component DBSs.

Sheth and Larson further divide the subcategory of FDBS into two types: loosely coupled and tightly coupled FDBS, based on who creates and maintains the federation and how the component databases are integrated. If the users themselves manage the federation, they call it a *loosely coupled* FDBS; whereas, if a global database administrator manages the federation and controls access to the component databases, the FDBS is *tightly coupled*. Both loosely coupled and tightly coupled FDBSs can support multiple federated schemata. However, if a tightly coupled FDBS is characterized by the presence of only one federated schema, it has a single federation.

The term *multidatabase* has been used by different authors to refer to different things. For example, W. Litwin et al. have used it to mean what Sheth and Larson call a loosely coupled FDBS. By contrast, Y. Breitbart and A. Silberschatz have defined multidatabase to be the tightly coupled FDBS of Sheth and Larson. Sheth and Larson have described additional, conflicting use of the term *multidatabase*.

The terms *loosely coupled* and *tightly coupled* FDBSs have also been used to distinguish between the degree to which users can perceive heterogeneity in an FDBS, among other factors. In this system of nomenclature (devised by this author and M.N. Kamel), a tightly coupled FDBS is characterized by the presence of a federated or global schema, which is not present in a loosely coupled FDBS. Instead of a global schema, loosely coupled FDBSs are integrated using other software, such as a user interface with a uniform "look and feel" or a standard set of queries used throughout the federation, thus contributing to a common operating environment.

In this case, the autonomous components of a loosely coupled FDBS are still cooperating to share data, but without a global schema. Thus, the users see only one DBS in a tightly coupled FDBS, whereas they are aware of multiple DBSs in the loosely coupled FDBS. Here, the tightly coupled FDBS obeys Date's rule zero, which states that to a user, a distributed system should look exactly like a nondistributed system.

Given this manner in which to characterize an FDBS, a *hybrid FDBS* is possible for which some of the component DBSs have a global schema that describe the data shared among them (tightly coupled), but other components do not participate in the global schema (loosely coupled).

An Expanded Taxonomy. An expanded taxonomy is proposed to provide a more comprehensive system to describe how databases are integrated, and to account for the perspectives of both the data administrator and the users. Essentially, most aspects of Sheth and Larson's taxonomy are logical and should be retained. However, instead of using Sheth and Larson's terms for tightly coupled federated database and loosely coupled federated database, the terms *tightly controlled* federated database and *loosely controlled* federated database, respectively, should be substituted.

This change focuses on the absence or presence of a central, controlling authority as the essential distinction between the two. In this case, the terms *tightly coupled* and *loosely coupled* can then be applied to describe how the user, rather than the data administrator, sees the federation. Given this change, the coupling between components in a federated database will describe how seamless and homogeneous the database looks to the users and applications.

The expanded taxonomy can accommodate federated databases that differ widely in their characteristics. For example, if a tightly controlled federated database is tightly coupled, the global data administrator and the global database administrator have exercised their authority and expertise to provide a seamless, interoperable environment that allows the federation's users to experience the illusion of a single database for their applications and ad-hoc queries.

A tightly controlled federated database can also be loosely coupled, in which case the global data administrator allows the users of the federation to see some heterogeneity with respect to the component databases.

Both conditions are within the realm of possibility. However, a loosely controlled federated database is almost certain to be loosely coupled. This is because a loosely controlled federated database lacks a central authority capable of mediating disputes about data representation in the federated schema and enforcing uniformity in the federation's interfaces to user applications. A loosely controlled federated database is not likely to be tightly coupled.

Local or Localized Schema vs. Component Schema vs. Export Schema. A local or localized database generally starts as a stand-alone, nonintegrated database. When a local, autonomous database is selected for membership in a federation, a local schema is defined as a conceptual schema of the component DBS that is expressed in the native data model of the component DBMS.

When the local database actually becomes a member of a federated database, it is said to be a *component database*. The schema associated with a given database component is called a *component schema*, which is derived by translating a local schema into the common data model of the FDBS. An *export schema* represents the subset of the component schema that can be shared with the federation and its users.

Similarly, Date defines a local schema as the database definition of a component database in a distributed database.

Federated Schema vs. Global Schema vs. Global Data Dictionary. A federated schema is an integration of multiple export schemata. Because the distributed database definition is sometimes called the global schema, federated schema and global schema are used interchangeably.

A global data dictionary is the same as a global schema that includes the data element definitions as they are used in the FDBS. A data dictionary is different from a schema, or database structure specification, because a data dictionary contains the definitions of attributes or objects, not just the configuration of tables, attributes, objects, and entities within that structure.

It is especially important to include the data element definitions with the export schemata when forming a federated database in which multiple data representations are likely. Simply having a collection of database structures is insufficient to complete a useful federated schema. It is necessary to know the meaning of each attribute or object and how it is construed in the component database.

Middleware vs. Midware. In a three-tier client/server architecture designed to connect and manage data exchange between user applications and a variety of data servers, the middle tier that brokers transactions between clients and servers consists of middleware, which is sometimes called midware.

P. Cykana defines middleware as a variety of products and techniques that are used to connect users to data resources. In his view, the middleware solution is usually devoted to locating and finding data rather than to moving data to migration environments.

In addition, Cykana describes two options for middleware. depending on the degree of coupling between the user and the data resource. Loosely coupled middleware products allow flexibility in specifying relationships and mappings between data items, whereas tightly coupled middleware products allocate more authority to standard interfaces and database administrators. Each option has its advantages and disadvantages, as follows:

- *Loosely coupled middleware*. This type of middleware does not require the migration or legacy data structures to be modified, but it allows users to access multiple equivalent migration systems transparently with one standard interface. Its disadvantage is that it does not prevent multiple semantics and nonstandard structures.
- *Tightly coupled middleware*. This option represents a more aggressive strategy that combines applications program interface (API) and graphical user interface (GUI) technologies, data communications, and data dictionary design and development capabilities to provide distributed data access. Data standardization and reengineering are required.

The concept of loose and tight coupling to middleware is somewhat similar to, but also differs slightly from, the loose and tight coupling between data resources as discussed by Sheth and Larson and other researchers. In the case of middleware, the coupling occurs between software at different tiers or layers (between the middle translation layer and the data servers); whereas, in the case of an FDBS, the coupling occurs between data servers that reside at the same tier. (However, this difference does not preclude software that achieves the coupling between data servers from being located in the middle tier.)

G.V. Quigley defines middleware as a software layer between the application logic and the underlying networking, security, and distributed computing technology. Middleware provides all of the critical services for managing the execution of applications in a distributed client/server environment while hiding the details of distributed computing from the application tier. Thus, midware is seen in a critical role for implementing a tightly coupled FDBS.

Similarly, Quigley considers middleware to be the key technology to integrate applications in a heterogeneous network environment.

Database Integration vs. Database Homogenization. Many organizations in both industry and government are interested in integrating autonomous (sometimes called "stovepipe") databases into a single distributed, heterogeneous database system. Many terms describe the various aspects of this integration. The multiplicity of terminology occurs because of the many ways in which databases can be integrated and because of the many simultaneous efforts that are underway to address integration problems.

Because the degree to which database integration takes place depends on the requirements of the organization and its users, the term *integration*, as it is used in various contexts, remains rather vague. For people whose fields of expertise are outside the realm of database technology, it is necessary to hide the specific details of database system implementation behind midware layers and a user interface that together create the illusion of a single, unified database. By contrast, more experienced users with knowledge of multiple DBMS can function efficiently in an environment that preserves some distinctions between the database components.

Within all architectural options, *database integration*, in its broadest sense, refers to the combination and transformation of database components into a database system that is homogeneous on at least one level (e.g., the data level, the schema level, the program interface level, or the user-interface level). Such an integrated database system must satisfy the primary goals of interoperability between database system components, data sharing, consistent data interpretation, and efficient data access for users and applications across multiple platforms.

K. Karlapalem et al. describe the concept of *database homogenization* as the process of transforming a collection of heterogeneous legacy information systems onto a homogeneous environment. Whereas they do not define what they mean by the term *homogeneous environment*, they list three goals of database homogenization:

- To provide the capability to replace legacy component databases efficiently
- To allow new global applications at different levels of abstraction and scale to be developed on top of the homogenized federated database
- To provide interoperability between heterogeneous databases so that previously isolated heterogeneous localized databases can be loosely coupled

This definition of database integration explicitly includes multiple architectures and implementations; by contrast, the description of database homogenization is associated with loose rather than tight coupling of localized databases into a homogeneous environment. Sometimes the term *database normalization* is used incorrectly to mean *database integration*.

Interoperability vs. Interoperation. The conditions necessary for interoperability include:

- Interconnectivity via the necessary networking facilities

- Resolution of system heterogeneity
- Resolution of semantic heterogeneity
- Derivation and integration of schemata and views

There are three levels of heterogeneity, including platform heterogeneity, data model heterogeneity, and semantic heterogeneity. Excluding semantic heterogeneity, the term system heterogeneity is seen to be some combination of platform heterogeneity (e.g., different DBMS software and implementation) and data model heterogeneity (e.g., schemata, query languages, integrity constraints, and nullness requirements). Because Karlapalem et al. have already listed the integration of schemata as an item separate from system heterogeneity, system heterogeneity logically should refer to the differences between DBMS vendors, transaction processing algorithms, query languages, query optimization techniques, integrity constraints, and nullness requirements. If this definition is assumed for system heterogeneity, the necessary conditions for database interoperability listed above become sufficient conditions.

Similarly, computer system heterogeneity and data management system heterogeneity must be resolved as a requirement for interoperability among existing information systems.

The achievement of database interoperability simply supplies users and applications with the ability to interoperate in a common data environment. It does not guarantee that interoperation will occur. Database interoperation results when users and applications take advantage of a common, integrated environment to access, share, and process data across multiple databases.

Legacy Information System vs. Migration Information System. Autonomous systems that become candidates for integration into a more modern, global, and distributed system sometimes have been called migration systems. These systems are supported by migration information systems with migration databases.

The term *migration databases* indicates unambiguously that the database in question has been chosen to be included in some form of a modern database system, especially a distributed system such as an FDBS. By contrast, the term *legacy information system* has been used in two different ways.

At one extreme, some people use legacy information system and legacy database to be synonymous with migration information system and migration database, respectively. Others have referred to a legacy information system as if it were not a migration information system and is therefore deliberately excluded from the final integrated database configuration. This is the opposite extreme.

More commonly than in the extreme cases, a subset of legacy data is deemed important to the users of a shared data resource. This means that some or all of the data in a legacy information system may be migrated during a database integration effort. For example, Cykana describes steps in the data integration process that start with the movement and improvement of data and progress to the shutdown of legacy systems. Karlapalem et al. refer to the difficulty of migrating legacy information systems to a modern computer environment in which some difference is presumed to exist between the legacy system and the modern system.

The author recommends that the following terminology be adopted as standard:

> *Legacy data* and *legacy information system* should refer to the original data and original format, as maintained in the original, autonomous information system before any modification or migration to a new environment has occurred. *Migration data* and *migration information system* should be used to describe the subset of the legacy data and software that has been chosen to be included into a new (and usually distributed) information resource environment. When data and software are modified to accommodate a new environment, they should be called migration instead of legacy.

TERMS ASSOCIATED WITH SEMANTIC HETEROGENEITY

Semantic heterogeneity refers to a disagreement about the meaning, interpretation, or intended use of the same or related data or objects. Semantic heterogeneity can occur either in a single DBS or in a multidatabase system. Its presence in a DBS is also independent of data model or DBMS. Therefore, the terminology associated with this problem is discussed in a separate section.

Semantic Interoperability vs. Database Harmonization

The terms *database integration* and *interoperability* were discussed previously in a general context. For distributed, heterogeneous database systems to be integrated in every respect, semantic heterogeneity must be resolved.

Problems associated with semantic heterogeneity have been difficult to overcome, and the terminology to describe semantic heterogeneity has evolved accordingly. For example, R. Sciore et al. define semantic interoperability as agreement among separately developed systems about the meaning of their exchanged data.

Whereas the exact meaning of the term *database harmonization* is not clear, one can infer that the goal of database harmonization must be related to providing an environment in which conflicts have been resolved between data representations from previously autonomous systems. This definition further implies that the resolution of semantic heterogeneity is a prerequisite for database harmonization.

Although a more precise definition of database harmonization is needed, it appears to be related to the idea of semantic interoperability.

Strong and Weak Synonyms vs. Class One and Class Two Synonyms

A synonym is a word that has the same or nearly the same meaning as another word of the same language. Because a metadata representation will include more attributes (e.g., data element name, type, length, range, and domain) than ordinary nouns, it was necessary to consider various levels of similarity and therefore, levels of synonymy.

M.W. Bright et al. have described the concept of strong and weak synonyms. Strong synonyms are semantically equivalent to each other and can be used interchangeably in all contexts without a change of meaning, whereas weak synonyms are semantically similar and can be substituted for each other in some contexts with only minimal meaning changes. Weak synonyms cannot be used interchangeably in all contexts without a major change in the meaning — a change that could violate the schema specification.

This concept is similar to one (introduced by the author and Kamel) that states that there are two classes of synonym abstraction: Class One and Class Two. Class One synonyms occur when different attribute names represent the same, unique real world entity. The only differences between Class One synonyms are the attribute name and possibly the wording of the definition, but not the meaning. By contrast, Class Two synonyms occur when different attribute names have equivalent definitions but are expressed with different data types and data-element lengths.

Class Two synonyms can share the same domain or they can have related domains with a one-to-one mapping between data elements, provided they both refer to the same unique real-world entity. The concept of a strong synonym is actually the same as that of a Class Two synonym because both strong synonyms and Class Two synonyms are semantically equivalent and they can be used interchangeably because they have the same data element type and length. By contrast, the concept of a Class Two synonym includes (but is not limited to) the concept of a weak synonym because the definition of a weak synonym seems to imply a two-way interchange in some contexts. The main difference is that the interchangeability of Class Two synonyms is determined not only by semantic context, but also by the intersection of their respective domains, as well as their data types and lengths.

Class Two synonyms allow for a one-way, as well as a two-way, interchange in some cases, whereas the "each-other" part in the definition of weak synonyms seems to preclude a one-way interchange. For example, a shorter character string can fit into a longer field, but not vice versa.

SUMMARY

This chapter presents a review of the rapidly growing vocabulary of database system technology, along with its conflicts and ambiguities. The solutions offered address some of the problems encountered in communicating concepts and ideas in this field.

This effort is intended to be a first step toward the development of a more comprehensive, standard set of terms that can be used throughout the industry. More work is needed to identify and resolve the differences in interpretation between the many terms used in data administration, database development, database administration, database research, and marketing as they occur in industry, government, and academia.

ACKNOWLEDGMENTS

This work was created by a U.S. government employee in the course of employment and is therefore in the public domain.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 1999	3. REPORT TYPE AND DATES COVERED Professional Paper
4. TITLE AND SUBTITLE A Review of Database System Terminology		5. FUNDING NUMBERS
6. AUTHOR(S) Dr. M. G. Ceruti		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Space and Naval Warfare Systems Center San Diego, CA 92152–5001		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12a. DISTRIBUTION/AVAILABILITY STATEMENTR Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Many publications, technical manuals, and marketing brochures related to databases originated from sources that exhibit a wide variety of training, background, and experience. Although the result has been an expanded technical vocabulary, the growth of standards, particularly with regard to a comprehensive, uniformly accepted terminology, has not kept pace with the growth in the technology itself. Consequently, the nomenclature used to describe various aspects of database technology is characterized, in some cases, by confusion and chaos. This is true for both homogeneous databases and for heterogeneous, distributed database systems.

The state of imprecision in the nomenclature of this field persists across virtually all data models and their implementations. The purpose of this chapter is to highlight some areas of conflict and ambiguity and, in some cases, to suggest a more meaningful use of the terminology.

Published in Handbook of Data Management, Ch. 2, S. Purba, Ed., CRC Press LLC, 1999.

14. SUBJECT TERMS Mission Area: Communications data base system terminology data item data element			15. NUMBER OF PAGES 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT Same as Report

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(include Area Code)*	21c. OFFICE SYMBOL
Dr. M. G. Ceruti	(619) 553-4068	Code D4221